Fall | 16

TFG

Alba Escobar Alberto

'FRONT-END' For Neuroimage Studies in Humans

Bachelor in Biomedical Engineering

Acknowledgments

Porque el camino acompañada es mucho más llano. Porque la vida teniendo personas que te quieren y te apoyan es mucho más vida.
Quiero agradecer a todos las personas que han hecho posible que hoy esté aquí, siendo quien soy, consiguiendo lo que he conseguido. En especial:

A mis padres, por su eterna ayuda.
A mis abuelos, por su eterno amor.
A mis hermanos, por ser mi mayor regalo.
A mi familia, por ser una familia unida, mi mejor abrigo, mi mejor hogar.
A mis amigas: Cande, Isa, Lucía y Eva, por ser las mejores compañeras de camino.
A Alberto, por ser mi gran apoyo.

Por último quiero agradecer a mi tutor, Manuel Desco, y a mi supervisor, David García, por su ayuda y disposición.

1 Table of Contents

2 Table of Figures

3 Introduction

3.1 Motivation

The Biomedical Imaging and Instrumentation Group (BIIG) of the University Carlos III de Madrid is a multidisciplinary team involved in research projects within different fields.
The lines of investigation in neuroimaging, which is one of their main focuses, draw into functional magnetic resonance imaging (fMRI), processing and analysis. The BIIG has developed specific and dedicated tools such as a pipeline or workflow[1] that enables the performance of a wide variety of neuroimaging processing methods.
Processing is crucial for the obtainment of interpretable and measurable results from raw data. Thus the aforementioned pipeline has become a valuable tool for the BIIG.

Researchers from the BIIG implemented the pipeline in Matlab and use it through a configuration file (Annex 13.1) composed of multiple command lines. Each application of the pipeline implies its own configuration file.
In general, one neuroimaging study requires the implementation of several pipelines and therefore the creation of several configuration files.
The development of multiple pipelines for the same study is an interesting approach in terms of study's robustness: either the application of different methods or the modification of the order of application of these methods may affect the results of fMRI analysis.

Prior to this Project, configuration files were coded line by line in a text editor using Matlab language. Therefore, the implementation was a difficult and time-consuming task that required programming skills.
The motivation for this Bachelor Thesis is the reduction of the complexity of this coding process and the reduction of the time and effort required for configuration files deployment

[1] Workflow: it is a way to name a group of processes that are performed one after the other.

3.2 Pipeline as a flux of different processes

The pipeline designed by the BIIG is an analysis strategy based on a multistep processing workflow. It merges in one single mechanism a broad range of existing transformation tools and methods in neuroimaging which are completely compatible among them and with the procedures carried out for structural processing. Therefore, the outcome from one method may become the input for the following, without order dependency.

The pipeline is implemented in Matlab and it is run through a configuration file written in Matlab language.

The structure of the workflow can be described using the following scheme:

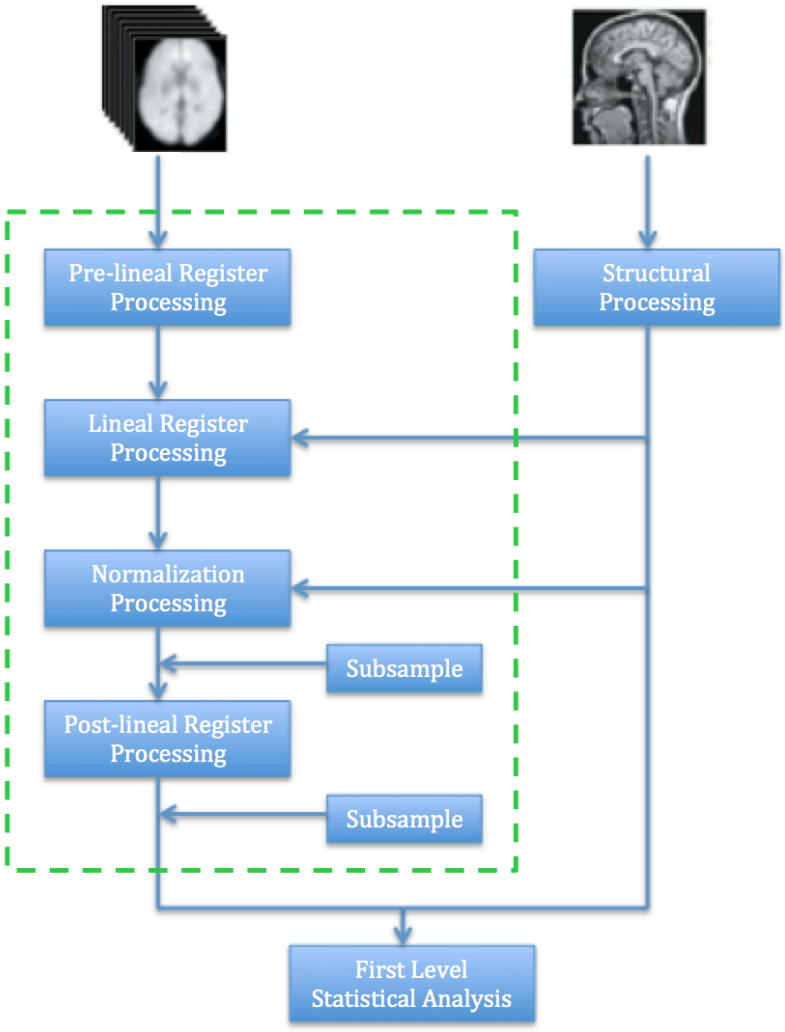


Figure 1 Overall Pipeline

Although the pipeline is constituted by a wide variety of processing steps, this bachelor thesis focuses on pre-processing steps and subsample processes (see green box in Figure 1)

Pre-processing steps can be classified attending to the order of application:

- Processes performed before lineal registration
- Process of lineal registration
- Process of spatial normalization
- Processes performed after lineal registration

Figure 2 depicts the processes of the pipeline included in this bachelor thesis and the sub-processes in which they are divided.

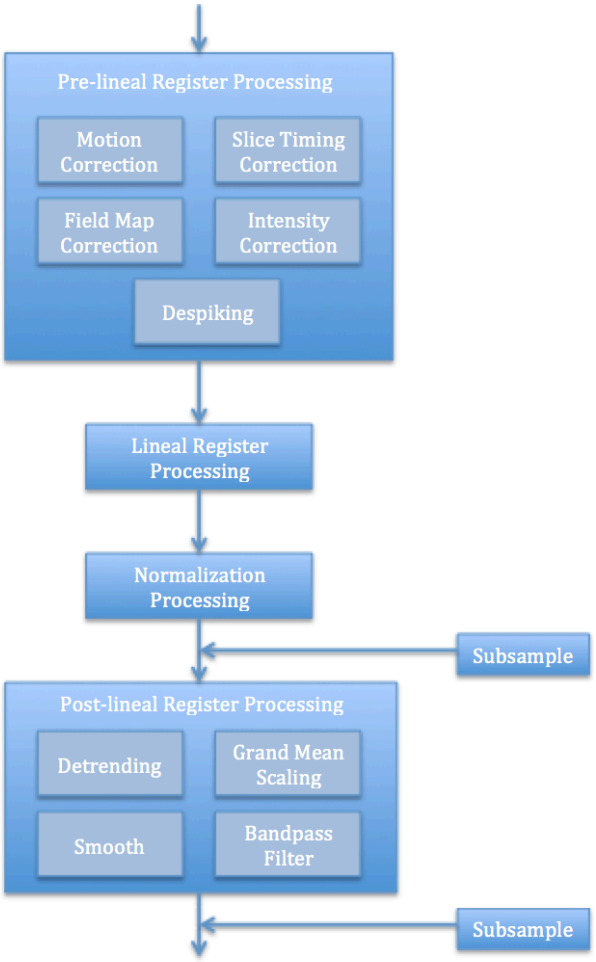


Figure 2 Parts of the Pipeline developed

All the processes in the previous figure are usually used in fMRI processing.

Next sections (3.2.1 and 3.2.5) provide a brief explanation of each one of the different processes.

3.2.1 Processes performed before lineal registration

- **Field Map Correction:** it corrects the BOLD images distortion arising from magnetic field inhomogeneity. It improves the quality of normalization and enhances the shape of the EPI signal.

- **Intensity Correction:** it reduces or removes the intensity artifacts in fMRI. These artifacts compromise success in image analysis [1].

- **Despiking:** This process automatically detects spikes in intensity, using Fourier series, and transforming those time series into more continuous time series [2]. In this way, the quality of the MRI is enhanced in terms of intensity inhomogeneity and motion distortion.

- **Motion Correction:** This method removes the noise arising from patient motion in MRI. There are three approaches to perform a motion correction:
    - Volume realignment
    - Out motion-related artifacts regression, through general linear models.
    - Motion confounded time points censoring [3].

- **Slice Timing Correction:** It adjusts the time course of voxels in each slice of the image by interpolation. It reduces the errors arising from the non-simultaneous acquisition of the 2D slices that form the final 3D volume. It enables further statistical models that assume the simultaneous acquisition of all voxels in the 3D volume [4].

3.2.2 Process of Lineal Registration

This procedure enables the adaptation of acquired functional MRI to the corresponding structural MRI in order to make them coincident. It improves the spatial normalization process and ROI analysis performance based on atlas regions [5].

3.2.3 Process of Spatial Normalization

Structural and functional images in MRI work in different spatial reference systems. Spatial normalization is an image registration method which moves the data from both images to a normalized space. It is essential for group analysis from voxel to voxel.

3.2.4 Processes performed after lineal registration

- **Smooth:** This technique removes high frequency information (small details) from the image, assuming that it is noise, and increasing the signal to noise ratio of the image.

- **Grand Mean Scaling:** It shifts all time series to the same scale through the normalization of the mean of all voxels [6].

- **Detrending** removes long-term drifts (low frequency drift), as for example baseline drifts. It emphasizes short-term drifts, and therefore, enhances weakly brain activation detection [7].

- **Bandpass Filtration:** This transformation preserves only a certain range of frequencies in the spectrum. In general, to implement this filter, two frequency parameters are specified:
    - Low cut-off frequency: below which the information is removed
    - High cut-off frequency: above which the information is removed

3.2.5 *Subsample processes*

This set of processes changes the voxel size of fMRI-processed images. Its implementation depends on the first level statistical analysis at which the images are later subjected.

There are two kinds of subsample processes in the workflow:

- **Subsample after Spatial normalization**
- **Subsample after pre-processing**

They differ on the application stage, despite their methodological basis are the same.

3.3 Tools used in the pipeline

The processes in the pipeline can be performed using one or several of the existing tools for neuroimaging processing. The tools and methods that the pipeline includes are listed below:

- **FSL:** FMRIB's software library that contains image analysis and statistical tools for functional, structural and diffusion MRI brain imaging data. FSL is available for Apple and PC (Linux) computers. Moreover, it is free for research purposes. [8]

- **SPM** (Statistical Parametric Mapping). It is an academic software analysis package that implements the theoretical concepts of statistical parametric mapping. It allows the analysis of brain functional imaging data sequences and it has been developed in MATLAB. [9]

- **AFNI:** (Analysis of Functional NeuroImages). It is a set of C programs for processing, analysing, and displaying fMRI data. It is implemented for mapping human brain activity. It is available for SGI, Solaris, Linux, and Mac OS X. Moreover, it is freely released for non-commercial use. [10]

- **ANTs:** (Advanced Normalization Tools). It is a toolkit that allows image registration and segmentation. It supports standardized multimodality image analysis. [11]

- **SINTEN**: procedure based on the study published by Zhexing Liu et al [12] to perform intensity correction.

- **MCFLIRT:** Intra-modal motion correction tool. It is used on fMRI time series. It is based on the optimization and registration techniques implemented in FSL. [13]

- **RORDEN**: tool that allows performing detrending and temporal filtration in fMRI. It was built using Matlab language and it is included in SPM8. [14]

- **DARTEL**: it is a tool included in SMP5 for spatial normalization

- **WAVELETS**: despiking algorithm contained in the BrainWavelet toolbox. It is written in Matlab and has releases for different operating systems (Mac OS X, Mavericks, Linux, Windows). [15]

- **ART:** toolbox from ArtRepair that allows post-processing of fMRI data and artifact detection. [16]

The following table shows which tools are permitted in the processes:

PROCESS	FSL	SPM	ANTS	AFNI	SINTEN	MCFLIRT	DARTEL	RORDEN	WAVELET	ART
Field Map Correction	✓	✓								
Intensity Correction					✓					✓
Despiking				✓					✓	
Motion Correction		✓	✓	✓		✓				
Slice Timing Correction	✓	✓								
Lineal Registration		✓	✓	✓						
Spatial Normalization	✓	✓	✓	✓			✓			
Smoothing	✓	✓		✓						
Grand Mean Scaling	✓	✓								
Detrending				✓				✓		
Bandpass FIlter				✓				✓		

3.4 State of the art

After an in-depth literature revision, we can find a variety of pipelines for fMRI processing. Understanding their foundations and basis is key for carrying out an appropriate work.

LONI pipeline, C-Pac Pipeline, DPARSF and Nypipe are four examples of pipelines with similar functionalities than the pipeline designed by the BIIG. In the following sections, a brief explanation of their contents and characteristics will be presented.

3.4.1 LONI Pipeline

The LONI Pipeline is a free application that allows the fast creation of process workflows. It enables the use of most available tools in neuroimaging without the request of external services.

This pipeline works on different operating systems: Mac Os X and Linux. Moreover it includes a simple user interface, focused on current research problems, and a pipeline library. The user interface is programmed in java and the library allows access to predefining neuroimaging solutions as modules, data and established workflows. [17]

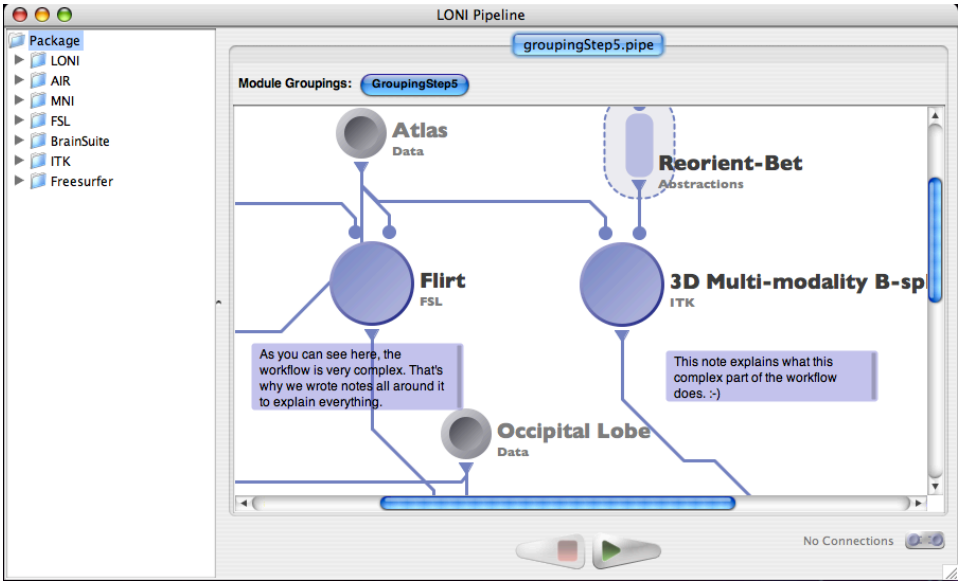


Figure 3 LONI User Interface

3.4.2 C-PAC Pipeline

It is an open-source software pipeline that allows automated pre-processing and analysis of resting-state fMRI data. It includes some of the existing tools for neuroimaging processing: AFNI, FSL and ANTS.

In addition, it presents a user friendly interface programmed in Python that allows both novice users and experts the access to the tools. Users can specify different preprocessing and analysis combinations, and they can create a wide variety of analysis pipelines to be run on an arbitrary number of subjects.

It does not support Windows but it works on Mac OS X and Linux. [18]

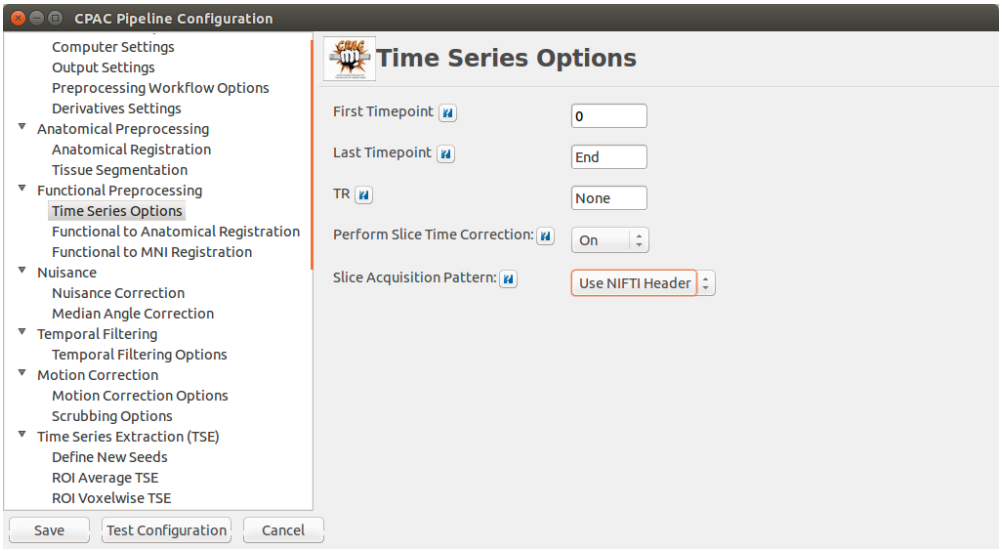


Figure 4 C-Pac User Interface

3.4.3 DPARSF Pipeline

Data Processing Assistant for Resting-State fMRI (DPARSF) is a Matlab toolbox based on SPM and REST. It allows creating workflows of processes for the analysis of resting-state fMRI. Apart, it has a user interface programmed in Matlab that eases the work of the researcher. Moreover, it includes popups that tell the user what will be done when selecting the different options.

It runs on any operating system. [19]

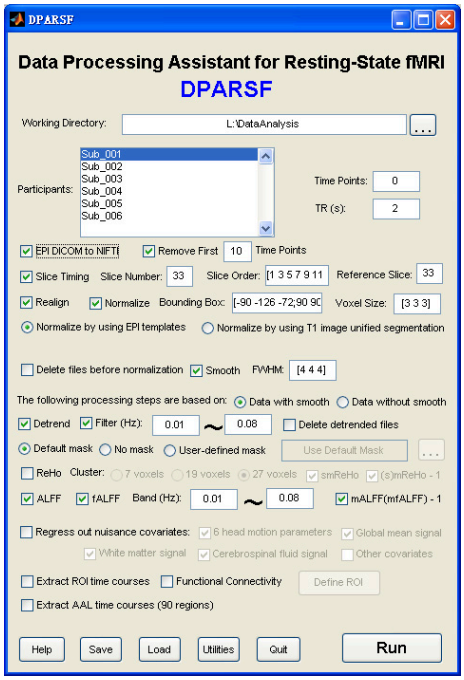


Figure 5 DPARSF User Interface

3.4.4 Nipype Pipeline

It is an open source pipeline developed by the community. It allows unifying all the existing interfaces for neuroimaging processing. Moreover, it enables the interaction of different tools in the same workflow. It includes ANTS, SPM, AFNI and FreeSurfer tools (among others).

The workflow is implemented in Python but it does not offer a user interface. [20]

3.4.5 Comparison

The previous examples have a similar function than the BIIG pipeline. All of them allow creating workflows and combine different of the existing tools for fMRI processing. Nevertheless, the majority of them include an advantageous feature that makes them user friendly. Despite they are written in different programming languages, they display a user interface that facilitates user-pipeline interaction.

The pipeline designed by the Group has not a user interface which hinders the activity of researchers.

This bachelor thesis aims to overcome this barrier and develop a graphical user interface.

4 Objective

4.1 Pipeline implementation.

The main objective is to create a graphical user interface, GUI, for pipeline implementation. This user interface has to **reach as many people as possible**. Also, it has to be **simple** to be used by people working on neuroimaging projects independently of their backgrounds. Besides it has to be **easy maintained** which implies a well-organized code structure. In order to fulfil the previous requirements, the application for the user interface has to be built attending to the specifications depicted in section 4.1.1.

4.1.1 Features

The application created for the user interface has to be:

- **Multiplatform:** it has to work in multiple operating systems (Windows, Mac Os X and Linux). In this way, the application could be used by researches despite the operating system they work with.

- **Open source:** the code is going to be available to the general public. Thus other developers might modify it or even improve it. The code would be universally accessible promoting the collaboration for among different researchers.

- **Well structured:** The source code should be structured in a well-defined order to allow further improvements. A proper organization of the code facilitates other developers understanding and continuing.

- **Free programming language:** it is an essential feature for the purpose of researchers' engagement. The deployment application in a private programming language prevents the access to it (due to money limitations or institution limitations).
- **User friendly:** the application has to should present an easy to manage and intuitive graphical user interface.

The pipeline will be used by people involved in neuroimaging research. Research teams are usually multidisciplinary and they involve people with many different background. Thus, every researcher should be able to understand and use the pipeline independently of his background and programming skills.

4.1.2 Scheme of the application

Apart from the previous features, the application should meet some specific requirements related to functionality. The GUI will be used for pipeline implementation and therefore it has to be constituted by the elements that allow pipelines creation:

- **Study selector:** user has to be offered with the possibility of selecting the amount and type of data (from fMRI studies) to be processed.

- **Process selector**: in order to select to which processing processes the data is going to be subjected.

  The user has to be provided with the possibility to select as many flows of processes as desired. Thus user can select performing:
  - A complete processing flow (using all the steps of the pipeline)
  - An incomplete processing flow (without using certain steps)
  - A combination of both (performing at the same time complete and incomplete flows).

  Moreover, different pathways can constitute the flows, as:
  - Certain steps of the pipeline can be performed using different tools at the same time;
  - The order of the steps can be varied.

  These alternatives will generate alternative fluxes or pipelines that have to be taken into account when building the GUI.

The combination of "n" selected studies and "m" selected processes will generate "n x m" different workflows as it is shown in the following figure:

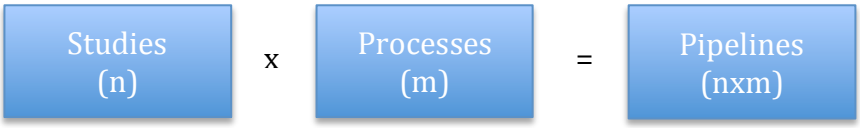


Figure 6 Workflow Generation

The possibility of generating "n x m" processes is an intrinsic characteristic of the GUI that differentiates this application from the existing ones.

4.1.3 Overall scheme

The application for the user interface is going to be integrated in a master workflow that can be described using the following scheme:

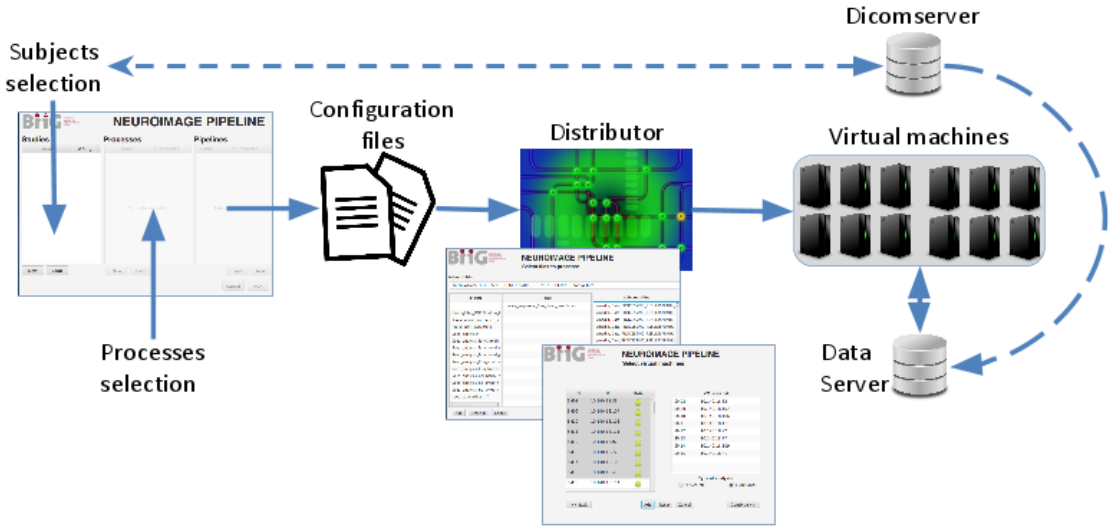


Figure 7 Overall Scheme

The overall workflow consists of four main blocks:

- **GUI (Graphical User Interface):** it comprises two different tools: a subject selector and a process selector. The subject selector is the one implemented by the student in this Bachelor Thesis. The functionality of both tools will be explained in advance.

- **Configuration files**: they are the final results from the user interface and they contain the command lines with the parameters of processes to be

performed. There can have as many configuration files as flows the user decide to perform at the same time.

- **Process distributor:** all the generated configuration files go to a process distributor, which shares out in different virtual machines. A member of the BIIG has already implemented this process distributor.

- **Virtual machines:** in them, the processing is carried out. The virtual machines retrieve images from data servers and process them in parallel way. It is possible because they contain the Matlab program that enables pipeline execution from configuration files.

6 Methods

6.1 Java, JavaFx

Once the objective is clear, I selected the proper programming language for the user interface development. After an in-depth analysis of different options, I decided to use Java.

Java is an open source and free programming language which includes a complete package for rich application creation called JavaFx. JavaFX has support for desktop computers and web browsers on Microsoft, Linux, and Mac [21,22]. This tool is called WORA (write once, run anywhere) [23] and it is crucial to make the application accessible from different platforms and from any operating system.

Before selecting java, other options were considered:

- Python: it was discarded as not everybody owns executor machine for Python applications
- C++: it was discarded because its framework for GUI creation is complex and tedious to use

6.2 Model-View-Controller

Javafx is suitable for using a programming pattern, called model-view-controller.

Model–view–controller (MVC) is a software architectural pattern or paradigms for the implementation of user interfaces [24]. It enables the deployment of an organized code and it facilitates the maintenance and expansion of the user interface. For these purposes, it divides a given software application into three parts that can be compared to the way information is provided to the user. Moreover, it states the interaction between them. The three parts conforming this paradigm are:

- **Controller:** it manipulates the model and updates it.

- **Model:** it saves the data that is brought back to the controller and is showed in the View. Whenever data change through the view, it is updated in the model by the controller.

- **View:** it is the visual representation of the model and user interface. The controller brings the data from the model and passes it to the view that presents it to users in an understandable format.

The following scheme summarizes the interactions between the three parts:

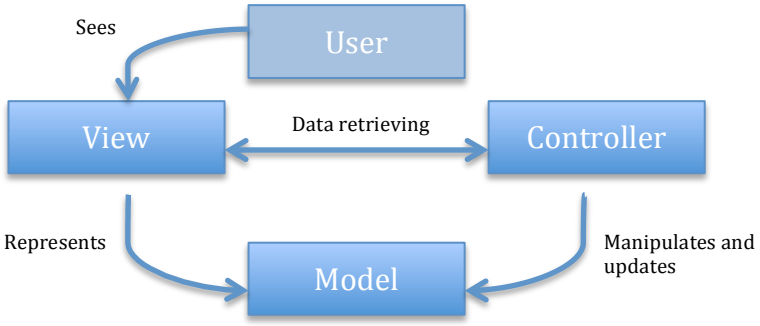


Figure 8 MVC Summary

6.3 Scene Builder

'JavaFX Scene Builder is a visual layout tool that allows users quickly designing JavaFX application user interfaces, without coding' [25]. By dragging and dropping different elements to a working panel, the developer can create the screens of the user interface. Once the views are designed and created, the tool generates automatically FXML files containing the code. These files are compatible with javaFx and can be joined to existing JavaFx projects, which contains the rest of the code (model and controller).

The following figure shows the interface of JavaFx Scene Builder where we can observe how a view has been built (the window inside the blue panel).

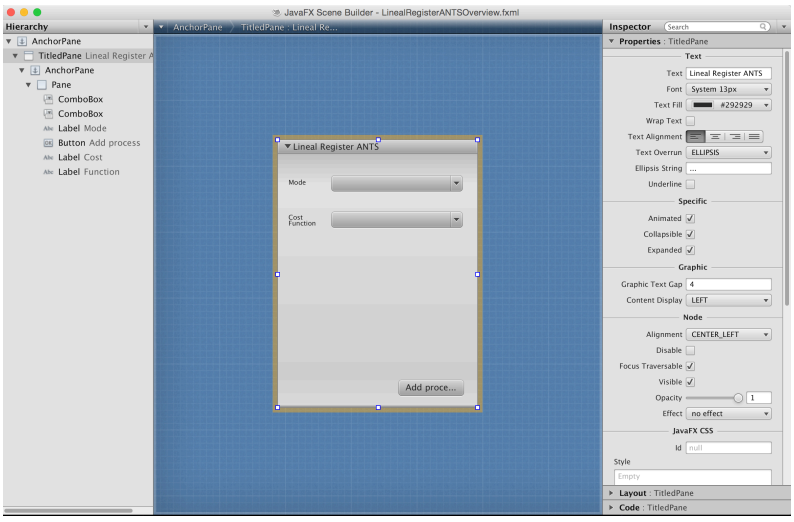


Figure 9 JavaFx Scene Builder Interface

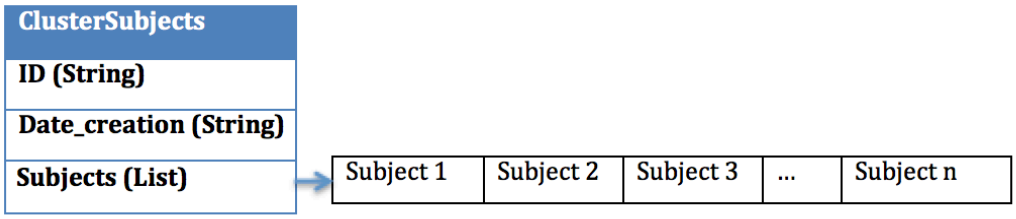
6.4 Pipeline Design

Once the programming environment and tools were defined, the designing stage started. In, this sense, two types of hierarchies were created. The use of these power structures allows organizing the information in our application. Moreover, they facilitate data management and support the developer in the application designing.

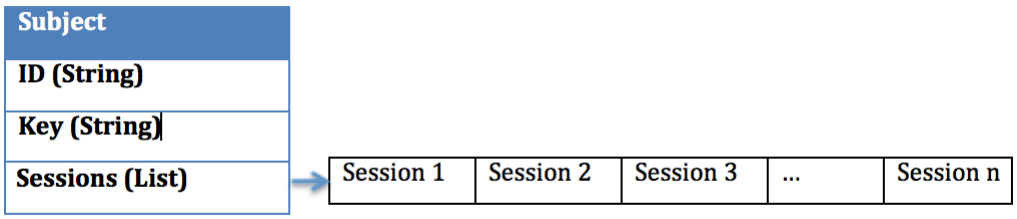
As stated before, the application should include a subject selector and a process selector. They must be classified as different elements because they deal with dissimilar information.
The hierarchies created for planning both designs will be explained in the following sections.

6.4.1 Study Selector Hierarchy

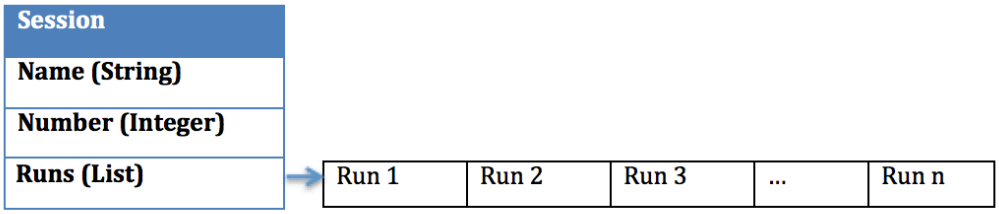
Study selector is the tool that allows selecting the data to be processed. In order to organize the data, I created a hierarchy. At the top of the hierarchy, it is the cluster of subjects, whose information included the following variables:



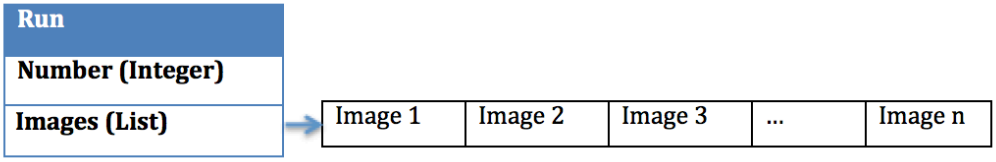
A cluster is composed of subjects. There can exist as many subjects as the user decide to study
The following variables constitute the information for each subject:



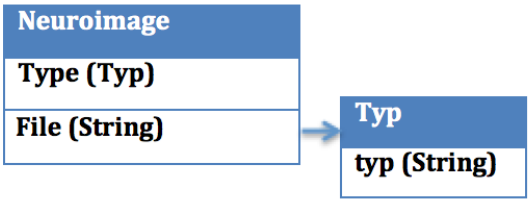
The data for the subjects is composed on sessions that are defined by the variables in the table below:



Different runs can constitute a sessions. The variables used for run definition are the following ones:



Finally, the runs are formed by neuroimages. A neuroimage is the core of the workflow and it is represented by the type and the name of the file:

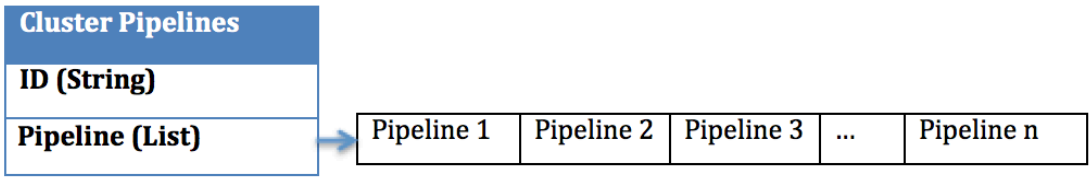


6.4.2 Process Selector Hierarchy

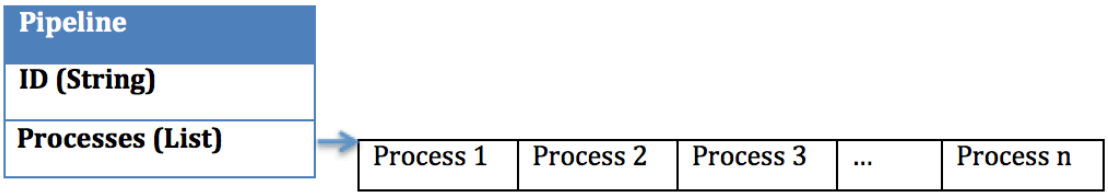
Process Selector is the tool that allows selecting the processes that form the pipeline and the order in which they are performed.

In order to organize the data and to facilitate application architecture, processes were arranged forming the following hierarchy.

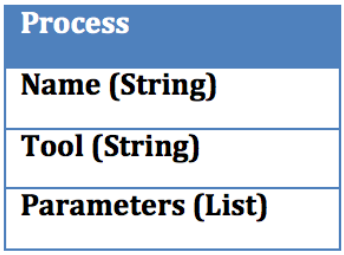
At the top of the hierarchy, the cluster of pipelines is:



The cluster of pipelines can be formed by a variety of pipelines. There may be as many pipelines as flows the user decides to perform. Each pipeline is defined by a variety of processes.



Each process is defined by the name of the tool employed to carry it out (Table 1) and a list of specific process parameters. In Annex 13.2, there are some tables that show the parameters defining each process.



6.4.2.1 Order Selection Function

The process selector tool provides the user with the possibility of selecting the order in which pre- and post-lineal register processes are performed, as there exist differences in the results when performing the processes on different order.

The order selection functions can be considered another type of processes and therefore, they are defined by specific parameters (Annex 13.3).

6.4.3 Process Selector. Specifications of its Design

Process Selector allows pipeline creation. Thus, this tool has to follow some specifications related to functionality:

1) A pipeline can be formed by a wide variety of procedures (See at section 3.2). Then, the tool should include them and let the user add them to the workflows.

2) The same process can be performed using different tools (See at table 1). Thus, the application should allow the user selecting the desired tool. If the user decided to use more than one tool for the same process, the application should create alternative workflows covering all alternative options.

3) The program should allow the user to avoid all kind of process. This option enables the comparison between performing a specific processing process and not-performing it.

6.5 Chronogram

Once the methods and tools for the pipeline implementation were determined, the work was planned. Prior to this bachelor thesis, the student did not have any Java programming skill. Regarding this drawback, time was distributed in the following way:

Periods (year 2015)	Tasks
February	Learning Period
March	Learning Period
April	Learning Period and Application Designing
May	Application Building
June	Application Building
July	Application Building
September	Work review and Essay Elaboration

6.5.1 Learning Period

The student underwent a long and intense learning period due to the lack of background in Java programming.. In this stage, the student learnt basic notions of java and javaFx package and practiced building simple applications and computing

solved examples. Moreover, these tasks were combined with lectures about other implemented pipelines to understanding the state of the art. In addition, the student analysed the documentation of the pipeline designed in the Group for better understanding their needs, requirements and potential solutions.

7 Results

7.1 Process Selector

The implementation of the pipeline requires a subject selection tool and a process selection tool and the designs of both tools have been explained previously, the design of the process selector is covered in this bachelor thesis, meanwhile another researcher from the BIIG implemented the subject selector.

7.2 Model-View-Controller Paradigm. Result of its implementation

The application for the pipeline was implemented following the model-view-controller paradigm. This paradigm allows building well-structured applications and it facilitates code expansion and modification.
In order to successfully obey the paradigm, the entire application was forced to fit the same pattern. Thus, the code composing each one of the views was divided in three fundamental parts:

- **Controllers**: they have to extend a class called AbstractController (Annex 13.4). This class is a pattern that was designed by an expert programmer of the Group. It contains all the necessary methods that enable controller-model and controller-view interactions.

- **Models:** they have to extend a class called AbstractModel (Annex 13.5). This class is a pattern designed by an expert programmer of the Group. It contains all the necessary methods that allow model-controller interaction.

- **Views:** they have not to extend any class. They consist on independent fxml files generated from JavaFx Scene Builder. They contain all the methods that allow view-controller interaction and physical view creation.

7.2.1 Model-View-Controller. Example: Motion Correction

The entire application comprises a total of 39 views. Each of these views is comprised by 3 pieces of codes (model, view, controller). Then, 117 code files constituted the view implementation.

The following examples match the three parts forming the implementation of the process “Motion Correction” performed using SPM tool.

7.2.1.1 Controller Example

```
public class MotionCorrectionSPMController extends AbstractController{

@FXML
private ComboBox<String> refVol;
@FXML
private ComboBox<Double> qual;
@FXML
private TextField separation;
@FXML
private ComboBox<Double> rtms;
@FXML
private Button ok;

private MotionCorrectionSPM motionCorrectionSPMModel;

public MotionCorrectionSPMController() {
}

/**Initialize method to load the combo boxes with the lists of possible options, define initial values
*and apply user's restrictions.
*/
@FXML
protected void initialize() {

ObservableList<String> volOptions = getVolReferences();
refVol.getItems().addAll(volOptions);
refVol.getSelectionModel().selectLast();

ObservableList<Double> qualityOptions = getQualityValues();
qual.getItems().addAll(qualityOptions);
qual.getSelectionModel().select(9);

separation.addEventFilter(KeyEvent.KEY_TYPED, Utils.restrictNoNumbers());
separation.setText("4");

ObservableList<Double> rtmOptions = getRTMValues();
rtms.getItems().addAll(rtmOptions);
}

/**
*Add as lists the options to the combo boxes
*/
private ObservableList<String> getVolReferences() {
ObservableList<String> vols= FXCollections.observableArrayList();
vols.add(MotionCorrectionFSL.FIRST);
```

```
vols.add(MotionCorrectionFSL.LAST);
vols.add(MotionCorrectionFSL.INTERMEDIATE);
vols.add(MotionCorrectionFSL.MEAN);

return vols;
}

// To add in a list the options to the combo boxes objects
private ObservableList<Double> getQualityValues() {
ObservableList<Double> quals= FXCollections.observableArrayList();
quals.add(0.0);quals.add(0.1);quals.add(0.2);quals.add(0.3);quals.add(0.4);
quals.add(0.5);quals.add(0.6);quals.add(0.7);quals.add(0.8);
quals.add(0.9);quals.add(1.0);

return quals;
}

// To add in a list the options to the combo boxes objects
private ObservableList<Double> getRTMValues() {
ObservableList<Double> rtms= FXCollections.observableArrayList();
rtms.add(1.0);rtms.add(2.0);

return rtms;
}

@Override
public void setModel(Model model) {
//Class conversion:
if(model instanceof MotionCorrectionSPM )
motionCorrectionSPMModel = (MotionCorrectionSPM) model;

else {
System.err.print("Cannot set model for MotionCorrectionSPMController" + model.toString()+" is
not a MotionCorrection SPM");
System.exit(1);
}
ok.setOnAction(new EventHandler<ActionEvent>() {

/**
*Define the events to occur when ok button is pressed
*/
public void handle(ActionEvent event) {   //Action event(clicked)
try{
motionCorrectionSPMModel.setParameters(refVol.getValue(),qual.getValue(), rtms.getValue(),
Integer.parseInt(separation.getText()));
//Print in console for testing
System.out.println(motionCorrectionSPMModel);}

//In case separation value is missed, the fiel will be filled with the initial value
catch(NumberFormatException e){
separation.setText("4");
motionCorrectionSPMModel.setParameters(refVol.getValue(),qual.getValue(), rtms.getValue(),
Integer.parseInt(separation.getText()));
System.out.println(motionCorrectionSPMModel);}
}});
}

/**
*To set the parameters in the model
```

```
*/
public void setModelParameters() {
refVol.getSelectionModel().select(motionCorrectionSPMModel.getRefVolStr());
qual.getSelectionModel().select(motionCorrectionSPMModel.getQuality());
separation.setText(Double.toString(motionCorrectionSPMModel.getSep()));
rtms.getSelectionModel().select(motionCorrectionSPMModel.getRtm());
}
}
```

7.2.1.2 Model Example

```
public class MotionCorrectionSPM extends AbstractModel {

public static final String FIRST = "First Volume";
public static final String LAST = "Last Volume";
public static final String INTERMEDIATE = "Intermediate Volume";
public static final String MEAN = "Mean Volume";
/**
* Associate values to each variable for posterior mapping
*/
public static final Map<String, Double> REFVOLMAP = new HashMap<String, Double>() {
private static final long serialVersionUID = 6531797824161961112L;
{
put(FIRST, 1.0);
put(MEAN, 0.0);
put(INTERMEDIATE, 0.5);
put(LAST, -1.0);
}
};

protected double quality;
protected double sep;
protected double rtm;
protected String refVolStr;
protected double refvol;
protected String mc;
protected String root;

public MotionCorrectionSPM() {
this(MEAN, 0.9, 2.0, 4);
}

public MotionCorrectionSPM(String refVolStr, Double qualDbl, Double rtmDbl,
double sepDbl) {
super("Motion Correction SPM", "");
setParameters(refVolStr, qualDbl, rtmDbl, sepDbl);
}
/**
* Set the values from the view to the variables of the model
*/
public void setParameters(String refVolStr, Double qualDbl, Double rtmDbl,
double sepDbl) {
refvol = REFVOLMAP.get(refVolStr);
this.refVolStr = refVolStr;

if (qualDbl != null) {
this.quality = qualDbl;
} else {
```

```
this.quality = 0.9;
}

if (rtmDbl != null) {
this.rtm = rtmDbl;
} else {
this.rtm = 0.9;
}

this.sep = sepDbl;
this.rtm = rtmDbl;
}
/**
* Get the current value for quality
*@return
*/
public double getQuality() {
return quality;
}
/**
* Set the value to model's variable quality
*/
public void setQuality(double quality) {
this.quality = quality;
}
/**
* Get the current value for sep
*@return
*/
public double getSep() {
return sep;
}
/**
* Set the value to model's variable sep
*/
public void setSep(double sep) {
this.sep = sep;
}
/**
* Get the current value for rtm
*@return
*/
public double getRtm() {
return rtm;}
/**
* Set the value to model's variable rtm
*/
public void setRtm(double rtm) {
this.rtm = rtm;}
/**
* Get the current value for refVolStr
*@return
*/
public String getRefVolStr() {
return refVolStr;}
/**
* Set the value to model's variable refVolStr
*/
public void setRefVolStr(String refVolStr) {
```

```
this.refVolStr = refVolStr;}
/**
* Get the current value for refVol
*@return
*/
public double getRefvol() {
return refvol;}
/**
* Set the value to model's variable refvol
*/
public void setRefvol(double refvol) {
this.refvol = refvol;}

//Create the string that set the parameters in the format of the configuration file
public String toCongifFile() {
this.root = super.toCongifFile();
this.mc = "\n" + root + "bool = 1;" + "\n" + root + "method = "
+ "'spm'" + ";" + "\n" + root + "refvol = " + refvol + ";"
+ "\n" + root + "sep = " + sep + ";" + "\n" + root + "rtm = "
+ rtm + ";" + "\n" + root + "quality = " + quality + ";" + "\n";

return mc;}

public String toString() {
return mc;}

//Associate the model with its controller and its view
public Controller initController() {
return (MotionCorrectionSPMController) Utils
.loadController(
"view/prelinealregister/motioncorrection/MotionCorrectionSPMOverview.fxml",this);}}
```

7.2.1.3 View

Views can be divided in two parts:

- FXML File: it is the file automatically generated from JavaFx Scene Builder. It contains the code for the view, which is compatible with the other two mentioned parts (model and controller) (Annex 13.6).
- Visual Component: it is the view created using JavaFx Scene Builder

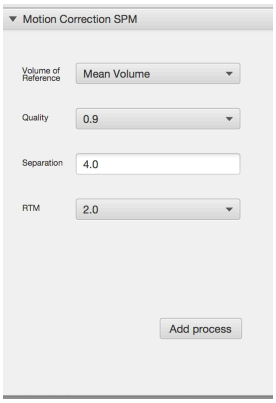


Figure 11 Motion Correction SPM View

7.3 Main Class

It is the class where the main method was implemented. The student chose the name of the class arbitrarily.

The main method is the one where the java program starts its execution.

Apart from the main method, this class contains methods for loading the panes forming the views and for loading the lists of objects containing the processes.

7.4 Utils

Apart from the files forming the views and the main class, the student created pieces of code for useful methods. Useful methods are defined as those resorted for a specific task in several parts of the code. By writing them in a file apart and calling it when needed, Java developer can save time and reduce the size and complexity of the code.

The table in Annex 14.7 shows a summary of the methods considered as utils when building the application.

7.5 Application Appearance

Once all codes in the application (models, controllers, views, main and utils) were created, they were merged in a single project to form the process selector. The aspect of the tool is shown in the following screenshots:

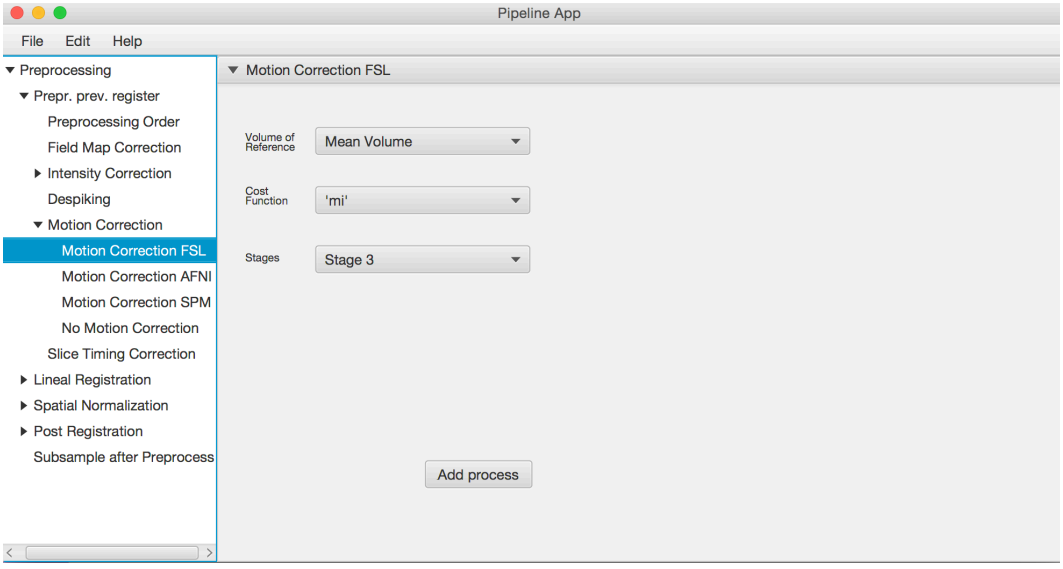


Figure 12 Application View 1

Two main parts form the application. In the left, there is a list with the processes that constitute the pipeline, and in the central part, the parameters defining each the process.

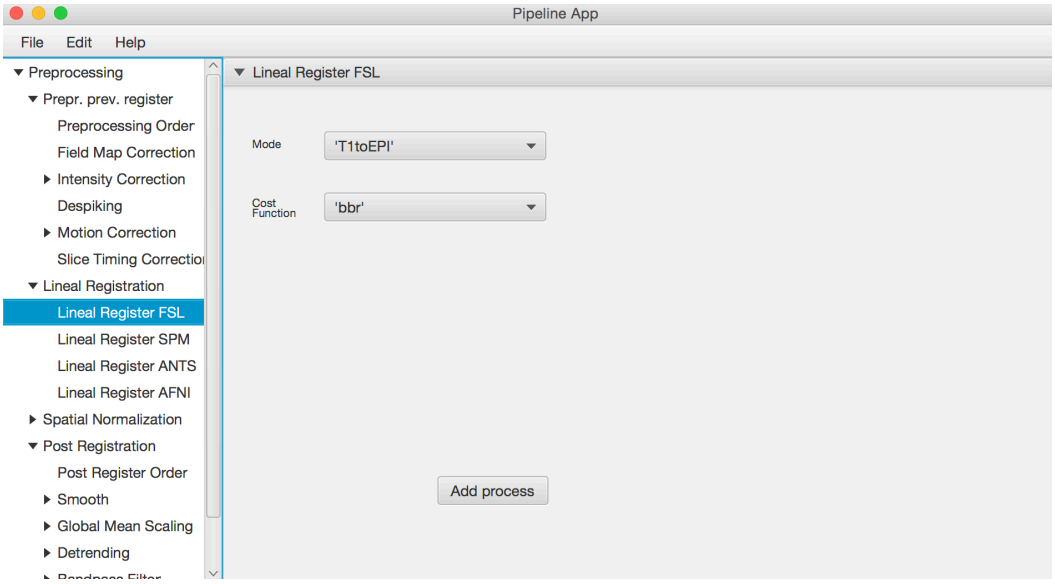


Figure 13 Application View 2

7.5.1 List of processes

In the right, there is the list of processes forming the pipeline. They have been shown as a tree view. The main branches of the tree corresponds to the first classification (section 3.2) and they divide the processes into:

- Pre-processes to registration
- Linear Registration
- Spatial normalization
- Post registration

These main branches can be unfolded to shown the second classification (sections 3.2.1-3.25). Finally, these branches can be unfolded to shown the tools available for each process.

7.5.2 Parameter Selector

Parameter Selector constitutes the central view and it offers the user a range of adjustable parameters.
The parameters appearing are specific of each process and tool. They can be modified by the user and set by pressing the button "Add process".

7.6 Application outcome

The outcome of the GUI has to be the configuration file required for pipeline implementation.
In order to create it, the parameters modified by the user through the GUI have to be printed in a text file. The structure followed to print them has to be equivalent to the one of the configuration file (Annex 13.1).

The mechanism to generate configuration files is based on the following statements:
1) Each time the user press the button "Add process", the process with its specific parameters have to be added to a configuration file.
2) If the added process is repeated, that is, the user adds the same process but with different parameters or performed with a different tool, other configuration file has to be created. This new configuration file has to contain the same information than the original one, but with the parameters of the repeated process.

7.7 Integration

The code developed for the process selector has to be merged with the one for the study selector (developed by another member of the research Group), as both tools are needed for a complete pipeline implementation.
In order to integrate these tools, a ''Master application'' was designed.

Figure 14 Master App Initial View

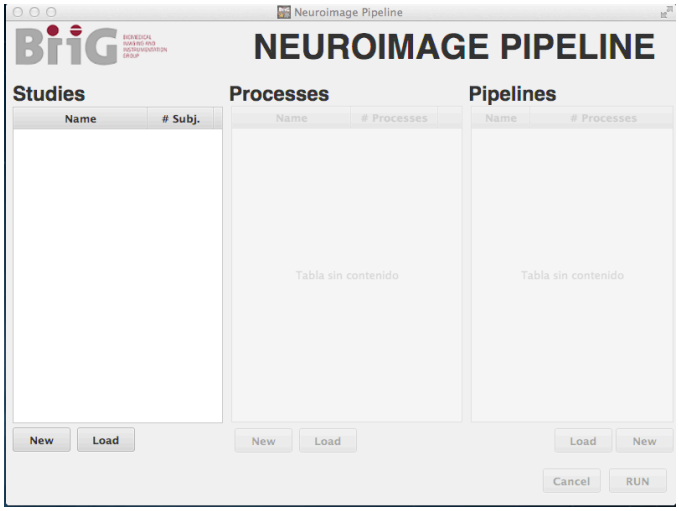


Figure 15 Master App View

This application allows joining the separated tools in a single one through XML files. Therefore, the process selector tool has to be adapted to generate XML files with outcome information.

7.7.1 Process Selector Adaptation

The adaptation of the process selector tool consists on creating an XML file from the results of the application.

The manipulation of the GUI outputs the lists of commands needed for the pipeline implementation. If the tool would not need integration, writing these lists of commands in texts files would be enough for application usability. However, the tool needs integration. For this purpose, a XML file compatible with the environment must be generated (Annex).

8 Socioeconomic Aspects and Regulatory Framework

8.1 Socioeconomic Aspects

The pipeline designed in the BIIG is a valuable tool for neuroimaging processing. Since its creation, it has been used in multiple studies. Not only in those headed by the BIIG, but in studies carried out by external institutions.

External studies have associated some expenses. Depending on the nature of the institution, these expenses vary. Moreover, the complexity of the studies influences directly the costs. The tariffs per hour of processing work are the following:

Type of Study	Public Institutions	Private Institutions
Simple Processing	€25/hr	€35/hr
Complex Processing	€50/hr	€90/hr

The creation of the GUI for pipeline implementation has eased and expedited processing work. Due to it, workflows are created easier and fast, and therefore, time expenditure and study costs have been reduced.

It is quite significant to decrease time consumption, because time economized from a work can be allocated to other lines of research. Moreover, it is also considerable to lessen study costs. On the one hand, the money saved from processing works can be invested in other research projects. On the other hand, processing work can be affordable for lower cost structure institutions.

8.2 Regulatory Framework

In the scope of this project, only free software tools were used.

On the one hand, for writing the code, java language was employed. Java is a free programming language and therefore, no regulatory aspect had to be taken into account. .

On the other hand, view's implementation was carried out on JavaFx Scene Builder. It is free software too, so no regulatory aspects had to be taken into account.

9 Future Work

There are two principal lines for future work. The first one is related to tool enhancement and the second one to tool integration.

9.1 Tool enhancement

The process selector can be enhanced in several ways:

- Creating views with tutorials. A future work will be adding tutorials that explain process basis and meaning of the parameters used. These tutorials can be added to each process constituting an excellent way of bringing the tool to students
- Adding pop-up windows to advice the user about application's functionality
- Tool Validation. Perform error-testing tasks to detect possible mistakes carried out by the student when she was developing the application.
- Adding a view to display the processes that have been added to the configuration file. In this way, the user can visualize the process he has added and can plan the processes to add.

9.2 Tool integration

As aforementioned, both the process selector and the study selector need to be integrated in a "master application". This integration requires that:

- The outcome of both applications are XML files
- The father application is developed
- The XML files and the "father" app are joined

So far, the first requirements have been fulfilled. However, the third requirement has not been implemented. Thus, further work should be carried out to join the XML files (the ones containing the outcomes of the process selector and the study selector) with the master application

10 Conclusion

To conclude, three fundamental aspects of this work has to be highlighted:

1) This bachelor thesis comprises the development of a functional application that facilitates the pipeline implementation. In this sense, the main objective of the work has been satisfied as the GUI has been implemented and serves its purpose.

2) The developed application fulfils the stated requirements:

- It has been created using a fixed pattern (Model-View-Controller). The code is easy to understand allowing code expansion and code maintenance.
- It has been built using a free programming language that works with all operating systems. The use of the application can be easily extended.
- The user friendly GUI empowers any researcher involved in neuroimaging to use the system despite his background.

3) The application has been built using java language. The student did not have any background in this programming language, so this bachelor thesis became a wonderful opportunity for her. She has expanded her programming skills and has learnt to interpret one of the most used languages.

12 References

[1] Vovk, U., Pernuš, F., & Likar, B. (2007). A review of methods for correction of intensity inhomogeneity in MRI. *Medical Imaging, IEEE Transactions on*, *26*(3), 405-421.

[2] Schomberg, H. (1999). Off-resonance correction of MR images. *Medical Imaging, IEEE Transactions on*, *18*(6), 481-495.

[3] http://fcp-indi.github.io/docs/user/motion.html

[4] Sladky, R., Friston, K. J., Tröstl, J., Cunnington, R., Moser, E., & Windischberger, C. (2011). Slice-timing effects and their correction in functional MRI. *Neuroimage*, *58*(2), 588-594.

[5] Zitova, B., & Flusser, J. (2003). Image registration methods: a survey. *Image and vision computing*, *21*(11), 977-1000.

[6] Dartmouth Brain Imaging Center Web Page. Available at: http://dbic.dartmouth.edu/wiki/index.php/Global_Scaling

[7] Tanabe, J., Miller, D., Tregellas, J., Freedman, R., & Meyer, F. G. (2002). Comparison of detrending methods for optimal fMRI preprocessing.*NeuroImage*, *15*(4), 902-907.

[8] Smith, S. M., Jenkinson, M., Woolrich, M. W., Beckmann, C. F., Behrens, T. E., Johansen-Berg, H., ... & Matthews, P. M. (2004). Advances in functional and structural MR image analysis and implementation as FSL. *Neuroimage*, *23*, S208-S219.

[9] Penny, W. D., Friston, K. J., Ashburner, J. T., Kiebel, S. J., & Nichols, T. E. (Eds.). (2011). *Statistical parametric mapping: the analysis of functional brain images: the analysis of functional brain images*. Academic press.

[10] Cox, R. W. (2012). AFNI: what a long strange trip it's been. *Neuroimage*, *62*(2), 743-747.

[11] Avants, B. B., Tustison, N., & Song, G. (2009). Advanced normalization tools (ANTS). *Insight J*, *2*, 1-35.

[12] Liu, Z., Wang, Y., Gerig, G., Gouttard, S., Tao, R., Fletcher, T., & Styner, M. (2010, March). Quality control of diffusion weighted images. In *SPIE medical imaging* (pp. 76280J-76280J). International Society for Optics and Photonics.

[13] Jenkinson, M., Bannister, P., Brady, M., & Smith, S. (2002). Improved optimization for the robust and accurate linear registration and motion correction of brain images. *Neuroimage*, *17*(2), 825-841.

[14] Chris Rorden's Neuropsychology Lab Web Page. Available at: http://www.mccauslandcenter.sc.edu/CRNL/clinical-toolbox

[15] Patel, A. X., Kundu, P., Rubinov, M., Jones, P. S., Vértes, P. E., Ersche, K. D., ... & Bullmore, E. T. (2014). A wavelet method for modeling and despiking motion artifacts from resting-state fMRI time series. *NeuroImage*, *95*, 287-304.

[16] Mazaika, P., Whitfield-Gabrieli, S., Reiss, A., & Glover, G. (2007, June). Artifact repair for fMRI data from high motion clinical subjects. In *annual meeting of the Organization for Human Brain Mapping*.

[17]Rex, D. E., Ma, J. Q., & Toga, A. W. (2003). The LONI pipeline processing environment. *Neuroimage*, *19*(3), 1033-1048.

[18] Sikka, S., Cheung, B., Khanuja, R., Ghosh, S., Yan, C., Li, Q., ... & Milham, M. (2014). Towards Automated Analysis of Connectomes: The Configurable Pipeline for the Analysis of Connectomes (C-PAC). In *5th INCF Congress of Neuroinformatics, Munich, Germany* (Vol. 10).

[19] Chao-Gan, Y., & Yu-Feng, Z. (2010). DPARSF: a MATLAB toolbox for "pipeline" data analysis of resting-state fMRI. *Frontiers in systems neuroscience*, *4*.

[20] Gorgolewski, K., Burns, C. D., Madison, C., Clark, D., Halchenko, Y. O., Waskom, M. L., & Ghosh, S. S. (2011). Nipype: a flexible, lightweight and extensible neuroimaging data processing framework in python. *Frontiers in neuroinformatics*, *5*.

[21] Gosling, J., Joy, B., Steele, G., Bracha, G., & Buckley, A. (2014). *The Java Language Specification*. Pearson Education.

[22] Morris, S. (2009). JavaFX in Action. Manning Publications Co.

[23] "Write once, run anywhere?". Computer Weekly. 2002-05-02. Retrieved 2009-07-27.

[24] Oracle Web Page: http://www.oracle.com/technetwork/java/javase/downloads/sb2download-2177776.html

[25] Reenskaug, T., & Coplien, J. (2013). More deeply, the framework exists to separate the representation of information from user interaction. *The DCI Architecture: A New Vision of Object-Oriented Programming*.

14 Annex

14.1 Example of a Configuration File

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
%Order of processes before lineal registration
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
%Order for the processes before registration.
%Icorrection always before motion correction if you choose remove volumes
opts.fmri.preproc.order{1}='icorr';
opts.fmri.preproc.order{2}='despiking';
opts.fmri.preproc.order{3}='slicet';
opts.fmri.preproc.order{4}='motion';
opts.fmri.preproc.order{5}='fieldmap';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
% Configuration of intensity correction
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
%Flag to perform process: 1 or 0 (yes or not)
opts.fmri.preproc.icorr.bool= 0;

% Method for this process: 'sinten' or 'art'
opts.fmri.preproc.icorr.method = 'sinten';

% Remove frames:1 or 0 (yes or not)
opts.fmri.preproc.icorr.outframes=0;

% Threshold for the process (3.5)
opts.fmri.preproc.icorr.alfa=3.5;

% Mask for the process: 'user' or 'thr'
opts.fmri.preproc.icorr.mask='thr';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
% Configuration of motion correction
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
%Flag to perform process: 1 or 0 (yes or not)
opts.fmri.preproc.motion.bool = 0;

%Method: 'fsl', 'ants', 'afni', 'spm'
opts.fmri.preproc.motion.method = 'spm';

%Reference volume (0-Mean from all 4D or number of volume for the 4D)
opts.fmri.preproc.motion.refvol = 0; % Number

%Cost function for the registration (FSL or ANTS): fsl: 'normcorr', 'mi', 'nmi', 'woods', 'corrratio', 'normi', 'leastsquares',
'msq'; ANTS:'pr', 'smi', 'cc', 'ssd','mi'
opts.fmri.preproc.motion.costfunction = 'normcorr';

%Separation parameter (only for SPM): i.e. 4;
opts.fmri.preproc.motion.sep = 4;

%RTM parameter (only for SPM): 1 or 2
opts.fmri.preproc.motion.rtm =2;

%Quality parameter 0.9 to 1 (Only for SPM)
opts.fmri.preproc.motion.quality =0.9;

%Number of stages: 3 or 4 (Only for FSL)
opts.fmri.preproc.motion.stages =3;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
% Configuration of despiking
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
%Flag to perform process: 1 or 0 (yes or not)
opts.fmri.preproc.despiking.bool=0;

%Method for the despiking:'afni' o 'wavelet'
opts.fmri.preproc.despiking.method = 'afni';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
% Configuration of slice timing correction
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
%Flag to perform process:1 or 0 (yes or not)
opts.fmri.preproc.slicet.bool = 0;

%Method for the despiking:'fsl' o 'spm'
opts.fmri.preproc.slicet.method = 'fsl';

%Adquistion mode: 'ascending','descending','interleaved_mt','interleaved_bu','interleaved_td'
opts.fmri.preproc.slicet.inter='interleaved_td'; % String adquisition mode

% Number of reference slice
opts.fmri.preproc.slicet.refslice=16;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
% Configuration of field map correction
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
%Flag to perform process:1 or 0 (yes or not)
opts.fmri.preproc.distc.bool =0;

%Method: 'fsl' 'ants' or 'spm'
opts.fmri.preproc.distc.method = 'spm';

%T1 or T2
opts.fmri.preproc.distc.mode='T1';

opts.fmri.preproc.distc.modul ='' ;

%Type of files: 'mag_phase_one_file_mag_phase_one_file' 'mag1_one_file_mag2_one_file_phase_one_file'
'mag1_one_file_mag2_one_file_phase1_one_file_phase2_one_file'
opts.fmri.preproc.distc.filetype = '';

%Files
opts.fmri.preproc.distc.FM{1} = '';
opts.fmri.preproc.distc.FM{2} = '';
opts.fmri.preproc.distc.FM{3} ='';
opts.fmri.preproc.distc.FM{4} ='';

%TEs for the fieldmaps
opts.fmri.preproc.distc.TE1 = '';
opts.fmri.preproc.distc.TE2 = '';

%Value of dwell
opts.fmri.preproc.distc.dwell='';

%String: y-
opts.fmri.preproc.distc.pol='y-';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
%%% Configuration parameters for lineal registration
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
%Flag no T1: 1 or 0 (yes or not)
opts.fmri.preproc.onlyEPI=0;

%Method for the registration: spm, fsl, ants, afni
```

```
opts.fmri.preproc.correg.method = 'fsl';

%Mode for the registration: 'T1toEPI' or 'T1toT2toEPI'
opts.fmri.preproc.correg.mode = 'T1toEPI';

%Cost function for the registration (FSL): 'mutualinfo', 'normmi', 'woods', 'corratio', 'normi', 'leastsq', 'labeldiff','bbr';
%Cost function for the registration (ANTS):'pr', 'smi', 'ssd','mi'
%Cost function for the registration (SPM):'nmi', 'mi', 'ecc','ncc'
opts.fmri.preproc.correg.costfunction = 'bbr';

%Separation for the registration (SPM) vector: default [4 2]
opts.fmri.preproc.correg.costfunction.sep = [4 2];



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
%%% Configuration parameters for spatial normalization
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
%Flag to perform process: 1 or 0 (yes or not)
opts.fmri.preproc.spnorm.bool=1;

%Mode for the registration: EPItoEPI T1toT1 or DARTEL
opts.fmri.preproc.spnorm.mode='T1toT1';

%Method for the registration: spm, fsl, ants afni
opts.fmri.preproc.spnorm.method = 'ants';

% 1 or 0 (yes or not) Normalize after GLM
opts.fmri.preproc.spnorm.post.bool=0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
%Subsample fmri after spatial normalization
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
%Flag to perform process: 1 or 0 (yes or not)
opts.preproc.subsample.postSN.bool=0;

%Size voxel
opts.preproc.subsample.postSN.vx=3;
opts.preproc.subsample.postSN.vy=3;
opts.preproc.subsample.postSN.vz=3;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
% Configuration parameters for processes post registration
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
%Order of processes after lineal registration
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
%Order for the processes after registration.
opts.fmri.preproc.order_post{1}='smooth';
opts.fmri.preproc.order_post{2}='gms';
opts.fmri.preproc.order_post{3}='detrend';
opts.fmri.preproc.order_post{4}='bpf';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
% Configuration of smooth
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
%Flag to perform process: 1 or 0 (yes or not)
opts.fmri.preproc.smooth.bool =1;

% Method: 'spm', 'fsl' o afni (Uniform smoothing)
opts.fmri.preproc.smooth.method = 'spm';

%Size of smooth for spm
opts.fmri.preproc.smooth.fwhm.x=12;
opts.fmri.preproc.smooth.fwhm.y=12;
opts.fmri.preproc.smooth.fwhm.z=12;

%Size of smooth for fsl or afni
opts.fmri.preproc.smooth.fwhm.g = 12;

%Flag to mask: 1 or 0 (yes or not)
opts.fmri.preproc.smooth.mask = 1;

%Type of mask: 'user', 'threshold', 'automask'
opts.fmri.preproc.smooth.maskt = 'user';

%Threshold value for threshold mask
opts.fmri.preproc.smooth.maskthreshold=0.8;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
% Configuration of grand mean scaling
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
%Flag to perform process: 1 or 0 (yes or not)
opts.fmri.preproc.gms.bool = 0;

% Method: 'spm', 'fsl'
opts.fmri.preproc.gms.method = 'fsl';

% Value for the gms:
opts.fmri.preproc.gms.val=10000;

%Flag for use of mask: 1 or 0 (yes or not)
opts.fmri.preproc.gms.mask = 0;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
% Configuration of detrending
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
%Flag to perform process: 1 or 0 (yes or not)
opts.fmri.preproc.detrend.bool = 1;

% Method: 'afni' 'Rorden'
opts.fmri.preproc.detrend.method = 'afni';

%Order for the trend:  0 lineal, 1 cuadratic, 2 cubic
opts.fmri.preproc.detrend.order=1;

%Flag for use of mask: 1 or 0 (yes or not)
opts.fmri.preproc.detrend.mask=0;

%For Rorden method
%Flag to use movement parameters: 1 or 0 (yes or not)
opts.fmri.preproc.detrend.realigparams=1;

%Flag to use wm: 1 or 0 (yes or not)
opts.fmri.preproc.detrend.wm=1;

%Flag to use csf: 1 or 0 (yes or not)
opts.fmri.preproc.detrend.csf=1;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
```

```
% Configuration of bandpass filter
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
%Flag to perform process: 1 or 0 (yes or not)
opts.fmri.preproc.bpfilter.bool = 1;

% Method: 'afni' 'DPARSF' 'Rorden'
opts.fmri.preproc.bpfilter.method = 'afni';

%Flag for use of mask: 1 or 0 (yes or not)
opts.fmri.preproc.bpfilter.mask=0;

%Low and High frequency for the filter (secs): vector
opts.fmri.preproc.bpfilter.freq = [0.008 100000];

%Flag for save the mean value: 1 or 0 (yes or not)
opts.fmri.preproc.bpfilter.save_mean=1;

%For afni method
%Flag to use automask: 1 or 0 (yes or not)
opts.fmri.preproc.bpfilter.automask=0;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
%Subsample fmri after preprocess
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
%Flag to perform process: 1 or 0 (yes or not)
opts.preproc.subsample.postpreproc.bool=0;

%Size voxel
opts.preproc.subsample.postpreproc.vx=3;
opts.preproc.subsample.postpreproc.vy=3;
opts.preproc.subsample.postpreproc.vz=3;
```

14.2 Parameters and Variables

Parameters defining each one of the processes and the variables that were used to name them.

Process of Intensity Correction

Type: Sinten	
Flag	**bool:** Boolean. 0 or 1
Outframes	**outframes:** Boolean. 0 or 1
Alfa	**alfa:** Double.
Mask	**mask:** String. 'thr' or 'user'

Type: Art	
Flag	**bool:**Boolean. 0 or 1
Outframes	**outframes:** Boolean. 0 or 1
Alfa	**alfa:** Double.
Mask	**mask:** String. 'thr' or 'user'

Motion correction

Type: fsl	
Flag	**bool:**Boolean. 0 or 1
Volume of reference	**refvol:** Integer. From 0 to 4
Cost Function	**costfunction:** String. 'normcorr', 'mi', 'nmi', 'woods', 'corrratio', 'normi', 'leastsquares' or 'msq'
Stages	**stages:** Ineger. 3 or 4

Type: ants	
Flag	**bool:** Boolean. 0 or 1
Volume of reference	**refvol:** Integer. From 0 to 4
Cost Function	**costfunction:** 'pr', 'smi', 'cc', 'ssd' or 'mi'

Type: afni	
Flag	**bool:** Boolean. 0 or 1
Volume of reference	**refvol:** Integer. From 0 to 4

Type: spm	
Flag	**bool:** Boolean. 0 or 1
Volume of reference	**refvol:** Integer. From 0 to 4
Separation	**sep:** Integer.
RTM	**rtm:** Integer. 1 or 2
Quality	**quality:** Double. From 0.9 to 1

Despiking

Type: afni	
Flag	**bool:** Boolean. 0 or 1

Type: wavelet	
Flag	**bool:** Boolean. 0 or 1

Slice timing correction

Type: fsl	
Flag	**bool:** Boolean. 0 or 1
Reference Slice	**refslice:** Integer.

Type: spm	
Flag	**bool:** Boolean. 0 or 1
Reference Slice	**refslice:** Integer.

Field map correction

Lineal registration

Type: spm	
Flag	**bool:** Boolean. 0 or 1
Mode	**mode:** String. 'T1toEPI' or 'T1toT2toEPI'
Cost Function	**costfunction:** String. 'nmi', 'mi', 'ecc','ncc'
Separation	**sep:** Integer Vector

Type: fsl	
Flag	**bool:** Boolean. 0 or 1
Mode	**mode:** String. 'T1toEPI' or 'T1toT2toEPI'
Cost Function	**costfunction:** String. 'mutualinfo', 'normmi', 'woods', 'corratio', 'normi', 'leastsq', 'labeldiff','bbr'

Type: ants	
Flag	**bool:** Boolean. 0 or 1
Mode	**mode:** String. 'T1toEPI' or 'T1toT2toEPI'
Cost Function	**costfunction:** String. 'pr', 'smi', 'ssd','mi'

Type: afni	
Flag	**bool:** Boolean. 0 or 1
Mode	**mode:** String. 'T1toEPI' or 'T1toT2toEPI'

Spatial normalization

Type: spm	
Flag	**bool:** Boolean. 0 or 1
Mode	**mode:** String. 'EPItoEPI', 'T1toT1' or 'DARTEL'
Normalize after GLM	**post.bool:** Boolean. 0 or 1

Type: fsl	
Flag	**bool:** Boolean. 0 or 1
Mode	**mode:** String. 'EPItoEPI', 'T1toT1' or 'DARTEL'
Normalize after GLM	**post.bool:** Boolean. 0 or 1

Type: ants	
Flag	**bool:** Boolean. 0 or 1
Mode	**mode:** String. 'EPItoEPI', 'T1toT1' or 'DARTEL'
Normalize after GLM	**post.bool:** Boolean. 0 or 1

Type: afni	
Flag	**bool:** Boolean. 0 or 1
Mode	**mode:** String. 'EPItoEPI', 'T1toT1' or 'DARTEL'
Normalize after GLM	**post.bool:** Boolean. 0 or 1

Subsample fmri after spatial normalization

Type: undefined	
Flag	**bool:** Boolean. 0 or 1
Vx	**vx: Double**
Vy	**vy: Double**
Vz	**vz: Double**

Smooth

Type: spm	
Flag	**bool:** Boolean. 0 or 1
X	**x: Double**
Y	**y: Double**
Z	**z: Double**
Mask Flag	**mask:** Boolean. 0 or 1
Mask	**maskt:** String. 'user', 'threshold' or 'automask'
Mask Threshold	**maskthreshold:** Double

Type: fsl	
Flag	**bool:** Boolean. 0 or 1

G	**g: Double**
Mask Flag	**mask:** Boolean. 0 or 1
Mask	**maskt:** String. 'user', 'threshold' or 'automask'
Mask Threshold	**maskthreshold:** Double

Type: afni	
Flag	**bool:** Boolean. 0 or 1
G	**g: Double**
Mask Flag	**mask:** Boolean. 0 or 1
Mask	**maskt:** String. 'user', 'threshold' or 'automask'
Mask Threshold	**maskthreshold:** Double

Global mean scaling

Type: spm	
Flag	**bool:** Boolean. 0 or 1
Val	**val:** Double
Mask	**mask:** Boolean. 0 or 1

Type: fsl	
Flag	**bool:** Boolean. 0 or 1
Val	**val:** Double
Mask	**mask:** Boolean. 0 or 1

Detrending

Type: afni	
Flag	**bool:** Boolean. 0 or 1
Order	**order:** Integer. From 0 to 2
Mask	**mask:** Boolean. 0 or 1

Type: rorden	
Flag	**bool:** Boolean. 0 or 1
Order	**order:** Integer. From 0 to 2
Mask	**mask:** Boolean. 0 or 1
Realigparams	**realigparams:** Boolean. 0 or 1
Wm	**wm:** Boolean. 0 or 1
Csf	**csf:** Boolean. 0 or 1

Bandpass filter

Type: afni	
Flag	**bool:** Boolean. 0 or 1
Mask	**mask:** Boolean. 0 or 1
Frequency	**freq:** Double Vector
Save_mean	**save_mean:** Boolean. 0 or 1
Automask	**automask:** Boolean. 0 or 1

Type: DPARSF	
Flag	**bool:** Boolean. 0 or 1
Mask	**mask:** Boolean. 0 or 1
Frequency	**freq:** Double Vector
Save_mean	**save_mean:** Boolean. 0 or 1

Type: Rorden	
Flag	**bool:** Boolean. 0 or 1
Mask	**mask:** Boolean. 0 or 1
Frequency	**freq:** Double Vector
Save_mean	**save_mean:** Boolean. 0 or 1

Subsample fmri after preprocess

Type: undefined	
Flag	**flag:** Boolean. 0 or 1
Vx	**vx: Double**
Vy	**vy: Double**
Vz	**vz: Double**

14.3 Parameters of Order Selection

Order for the processes before registration

Type: undefined	
First process	**String: 'icorr', 'despiking', 'slicet', 'motion', 'fieldmap'**
Second process	**String:** 'icorr', 'despiking', 'slicet', 'motion', 'fieldmap'
Third process	**String:** 'icorr', 'despiking', 'slicet', 'motion', 'fieldmap'
Fourth process	**String:** 'icorr', 'despiking', 'slicet', 'motion', 'fieldmap'
Fifth process	**String:** 'icorr', 'despiking', 'slicet', 'motion', 'fieldmap'

Order for the processes after registration

Type: undefined	
First process	**String:** 'smooth', 'gms', 'detrend', 'bpf'
Second process	**String:** 'smooth', 'gms', 'detrend', 'bpf'
Third process	**String:** 'smooth', 'gms', 'detrend', 'bpf'
Fourth process	**String:** 'smooth', 'gms', 'detrend', 'bpf'

14.4 Abstract Controller

```
package pipeline.utils;
import javafx.fxml.FXML;
import javafx.scene.Node;


/**
 * This class abstracts a generic controller bundle to an fmxl file
 * The extended classes implementing this one are bundle to the graphic part within
 *  Instanced controller {@link Node}
 *
 *  Any instance of this abstract class must implements the Controller interface.
 *  This way the controller first initialize the {@link Node} and after actions to the {@link Model}
 *  are bundled other way JavaFx would crash!!!
 *
 *  It's highly recommended instanced this abstract class using {@link Utils#loadController(String,
Model)}
*
 */
public abstract class AbstractController  implements Controller {
	@FXML
	private Node view;


	public Node getView() {
		return view;
	}

	public void setView(Node view) {
		this.view = view;
	}

	@FXML
	protected abstract void initialize();

	public void setModelParameters() {}

}
```

14.5 Abstract Model

```
package pipeline.utils;
import java.util.ArrayList;
import java.util.List;
import org.apache.commons.lang3.builder.HashCodeBuilder;
import javafx.scene.control.TreeItem;

public abstract class AbstractModel extends TreeItem<String> implements Model,
Comparable<AbstractModel>,Cloneable{
protected final Controller controller;
protected final String configFileString;


public AbstractModel(String treeNodeName,String configFileString) {
super(treeNodeName);
this.configFileString = configFileString;
controller = initController();
}

/**
*
* @param treeNodeName
*/
public AbstractModel(String treeNodeName) {
this(treeNodeName,treeNodeName);
}

/**
*
* @return the controller bundled to the model
*/
public abstract Controller initController();

public Controller getController() {
return controller;
}

/**
* Recursively get the from the root the config line
* @return
*/
public String toCongifFile() {
String res= configFileString;
AbstractModel parent = getParentModel();
while(parent != null) {
res = parent.configFileString.concat(".").concat(res);
parent = parent.getParentModel();
}
return res;
}


private AbstractModel getParentModel() {
TreeItem<String>  p= this.getParent();
if(p != null && p instanceof AbstractModel)
return (AbstractModel)p;
return null;
}
```

```
public String getConfigFileString() {
return configFileString;
}

@Override
public int compareTo(AbstractModel o) {
if(!getClass().getName().equals(o.getClass().getName())) //Check are the same class
return Integer.MAX_VALUE;

return getModelSpecifics().compareTo(o.getModelSpecifics());
}
@Override
public AbstractModel clone() {
try {
return (AbstractModel)super.clone();
} catch (CloneNotSupportedException e) {
e.printStackTrace();
return null;
}
}

@Override
 public int hashCode() {
 return new HashCodeBuilder(17, 31).append(getModelSpecifics()).toHashCode();
 }

@Override
public boolean equals(Object model) {
if (!(model instanceof AbstractModel))
return false;
if (model == this)
return true;
return compareTo((AbstractModel) model) == 0 ? true:false;
 }


protected String getModelSpecifics() {
List<String> specs = new ArrayList<String>();
String[] byCarrigeRet =  toCongifFile().split("\n");
for(String spec: byCarrigeRet)
if(spec.length() > 0) {
String[] byRoot = spec.split("\\.");
specs.add(byRoot[byRoot.length -1 ]);
}

return String.join("\n", specs);
}
}
```

14.6 FXML File

```
<?xml version="1.0" encoding="UTF-8"?>

<?import java.lang.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.*?>

<AnchorPane xmlns:fx="http://javafx.com/fxml/1" xmlns="http://javafx.com/javafx/2.2"
fx:controller="pipeline.view.prelinealregister.motioncorrection.MotionCorrectionSPMController">
 <children>
  <TitledPane text="Motion Correction SPM" AnchorPane.bottomAnchor="0.0"
AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">
    <content>
     <AnchorPane prefHeight="380.0" prefWidth="300.0">
      <children>
       <Pane prefHeight="376.0" prefWidth="355.0" AnchorPane.bottomAnchor="0.0"
AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">
         <children>
          <ComboBox fx:id="refVol" layoutX="80.0" layoutY="30.0" prefHeight="25.0"
prefWidth="200.0" />
          <ComboBox fx:id="qual" layoutX="80.0" layoutY="85.0" prefHeight="25.0"
prefWidth="200.0" />
          <TextField fx:id="separation" layoutX="80.0" layoutY="140.0" prefHeight="25.0"
prefWidth="200.0" promptText="Only integer values" />
          <ComboBox fx:id="rtms" layoutX="80.0" layoutY="195.0" prefHeight="25.0"
prefWidth="200.0" />
          <Label layoutX="15.0" layoutY="33.0" text="Volume of ">
           <font>
            <Font size="10.0" fx:id="x1" />
           </font>
          </Label>
          <Label font="$x1" layoutX="15.0" layoutY="42.0" text="Reference" />
          <Label font="$x1" layoutX="15.0" layoutY="90.0" text="Quality" />
          <Label font="$x1" layoutX="15.0" layoutY="145.0" text="Separation" />
          <Label font="$x1" layoutX="15.0" layoutY="200.0" text="RTM" />
          <Button fx:id="ok" layoutX="182.0" layoutY="340.0" mnemonicParsing="false"
prefHeight="25.0" prefWidth="100.0" text="Add process" />
         </children>
        </Pane>
       </children>
      </AnchorPane>
    </content>
  </TitledPane>
 </children>
</AnchorPane>
```

14.7 Table with Utils methods

Method	Description
loadController	For model controller association
restrictToNumbers	To restrict typing to integers in the GUI
restrictToDoubles	To restrict typing to doubles in the GUI
printTree	To check the list of processes conforming the pipeline in console
checkDuplicate	To check element duplication in the different components of the cluster of piles
addPoint	To built the command lines that formed the outcome file